POLITECNICO DI TORINO
Repository ISTITUZIONALE

Implementation of the TCG DICE Specification into the Keystone TEE Framework

Original
Implementation of the TCG DICE Specification into the Keystone TEE Framework / Bravi, E., Sisinni, S., Ferro, L., Donnini, V., Lioy, A.. - In: IEEE ACCESS. - ISSN 2169-3536. - 13:(2025), pp. 142284-142303. [10.1109/ACCESS.2025.3596829]

Availability:
This version is available at: 11583/3002386 since: 2025-08-11T14:38:50Z

Publisher:
IEEE

Published
DOI:10.1109/ACCESS.2025.3596829

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

18 November 2026

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 7 July 2025, accepted 30 July 2025, date of publication 7 August 2025, date of current version 18 August 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3596829

RESEARCH ARTICLE

Implementation of the TCG DICE Specification Into the Keystone TEE Framework

ENRICO BRAVI, (Student Member, IEEE), SILVIA SISINNI, (Member, IEEE), LORENZO FERRO, VALERIO DONNINI, AND ANTONIO LIOY

Dipartimento di Automatica e Informatica (DAUIN), Politecnico di Torino, 10129 Turin, Italy

Corresponding author: Enrico Bravi (enrico.bravi@polito.it)

This work was supported in part by the Secure Platform for Information and Communications Technology (ICT) Systems Rooted at the Silicon Manufacturing Process (SPIRS) Project through the European Union's Horizon 2020 Research and Innovation Program under Grant 952622, in part by the Project SEcurity and RIghts in the CyberSpace (SERICS) through the National Recovery and Resilience Plan (NRRP) MUR Program funded by the European Union-NextGenerationEU under Grant PE00000014, and in part by the Project PNRR-NGEU (Piano Nazionale di Ripresa e Resilienza - Next Generation EU) under Grant MUR–DM 352/2022.

ABSTRACT The rapid adoption of IoT devices introduced significant security challenges because they are often resource-constrained and operate in untrusted environments. Their adoption into critical scenarios makes it paramount they remain trustworthy. A possible solution is the Trusted Execution Environment (TEE), which isolates and protects sensitive code and data in use. Many TEE implementations exist (e.g., ARM TrustZone), yet most are closed-source, with specific hardware requirements. To overcome these issues, open-source solutions like Keystone have been proposed. Keystone is a framework for building customizable TEEs targeting RISC-V devices, based on the Physical Memory Protection security extension. Enforcing local protection must be coupled with the ability to verify the device is behaving as intended. This can be achieved with attestation techniques, but for the highest security level, some additional components are required. While Keystone defines basic requirements for attestation, it does not support architectures based on standard specifications. The Trusted Computing Group developed the Device Identifier Composition Engine (DICE) specifications to establish strong identity and integrity for IoT devices. In this paper, we propose the DICE integration in Keystone, to support secure boot and attestation. We detail the design, implementation, and evaluation of this solution, demonstrating its effectiveness in both security and performance.

INDEX TERMS IoT, physical memory protection, remote attestation, RISC-V, root of trust, secure boot, trusted execution environment.

I. INTRODUCTION

The Internet of Things (IoT) has experienced remarkable growth in recent years [1], with the number of connected devices increasing significantly. In 2024, the number of globally connected IoT devices was estimated to be around 18.8 billion, and statistical projections indicate that this number will nearly double, reaching over 32.1 billion by 2030 [2]. This exponential growth underscores the increasing integration of IoT devices into daily life, including safety-critical sectors. Smart-X paradigms (e.g. Smart Factory, Smart City, Smart Grid, Smart Transportation) heavily rely on the widespread and ubiquitous presence of IoT devices that continuously collect data from the environment (Figure 1) to create sophisticated user services and optimised industrial processes. However, many IoT devices are placed in easily accessible locations, which gives opportunities to attackers for direct manipulation, such as a malicious firmware update. As a consequence, these devices require robust security measures to protect their operations. However, this contrasts with other typical requirements, such as small size, low power consumption, low cost, and fast time-to-market, making implementing strong security measures challenging.

A Trusted Execution Environment (TEE) is an effective technology to enhance the security and reliability of IoT devices. A TEE is a protected execution area inside the

The associate editor coordinating the review of this manuscript and approving it for publication was Alessandro Pozzebon.

© 2025 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License.
For more information, see https://creativecommons.org/licenses/by/4.0/



FIGURE 1. IoT devices use cases.

platform, where sensitive code and data are stored and processed, protected from an attacker who has managed to gain kernel-level privileges on the main system. Thus, TEEs prevent malicious applications from stealing sensitive and valuable data or altering the processing of high-risk data. The implementation of TEEs varies greatly depending on the underlying technology. For example, some TEEs are realised by adding new states to the main Central Processing Unit (CPU) execution mode, e.g. ''secure'' and ''non-secure'', with a trusted software monitor to manage the switching between the two modes, that share the same computational resources. Other TEEs rely on executing secure applications in a hardware partition physically separated from the main processor. Many major TEE implementations, typically those coming from the industry, are proprietary and closed-source (e.g. Intel Trust Domain Extensions (TDX) [3], Qualcomm Secure Execution Environment (SEE) [4], Huawei iTrustee [5]) while those proposed by the academia or standardization bodies are open-source (e.g. Open Portable Trusted Execution Environment (OP-TEE) [6], Komodo [7], Sanctum [8]) so they can be extended and modified according to needs. This great variety of implementations originates differences in the offered security properties, making each implementation tied to specific contexts and use cases. Keystone, proposed in 2020 by Lee et al. [9], is an open-source framework for building customizable TEEs. The idea behind Keystone is not to be a ''finished'' TEE, with pre-established security features, but to offer a framework for TEE development, tailored to specific use cases and contexts. Keystone targets RISC-V, an open-source, royalty-free Instruction Set Architecture (ISA) which is emerging as a promising alternative to ARM for future IoT devices, due to its great flexibility and lack of commercial restrictions. For this reason, research on RISC-V based TEEs, such as Keystone, is a strategic and attractive choice, as it is likely to play a crucial role in the evolution of future IoT devices.

Locally enforcing security is not always enough. For this reason, it can be necessary to also verify a device is behaving as intended. Remote Attestation (RA) techniques can be used to remotely verify if an IoT device is in a trustworthy state. RA allows a remote entity to gain assurance about the trustworthiness of a device before interacting with it. This is achieved by requesting cryptographically secured proof regarding the state of a device and comparing it with expected values. This is typically done periodically to continuously assess the device's trustworthiness. The security of RA is based on the presence of specific security capabilities that create a Root of Trust (RoT). This component aims to provide strong identity and critical security features for device integrity verification. On PCs and server platforms, the RoT used to enable RA is typically the Trusted Platform Module (TPM), implemented mostly as a discrete chip to provide a Hardware Root of Trust (HRoT), hosted on the platform's motherboard. However, the TPM is often impractical or infeasible to integrate into IoT and embedded systems for several reasons, such as space, cost, and power consumption. Thus, the Trusted Computing Group (TCG) has proposed a new specification, the Device Identifier Composition Engine (DICE) [10], which defines a HRoT for resource-constrained devices.

Keystone supports a default integrity verification process for critical applications in its firmware, the Security Monitor (SM), which, however, does not comply with the standard specifications. Implementing DICE in Keystone enhances the security of resource-constrained devices by adding efficient and versatile methods to verify hardware and software identity. Furthermore, it can be easily integrated into existing frameworks and security protocols.

In this paper, we propose an integration of DICE into the Keystone framework. In particular, we present the design and implementation of the HRoT, based on the TCG DICE specification. The aim is to leverage DICE to provide strong identity and secure integrity verification capabilities to Keystone-enabled devices. Our implementation enables attestation of launched enclaves during device operation in a standardised format, using a DICE-compliant RoT and the standardised DICE specification. In this sense, our approach supports a form of ''continuous verification'' of the device's state, allowing regular verification of active enclaves' integrity. We detail the architecture design and implementation, and evaluate our proposal's performance by comparing it with the original framework, demonstrating its effectiveness and efficiency in enhancing the security of IoT deployments.

Paper Structure The paper is structured as follows. Section II describes the fundamental concepts and technologies on which this work is based. Section III details the contributions of our work, clearly highlighting the advances over the default Keystone implementation. Section IV mentions and briefly describes some related work. Section V presents the design of the proposed architecture and provides implementation details. Section VI analyses the security properties of the proposal. Section VII presents the tests

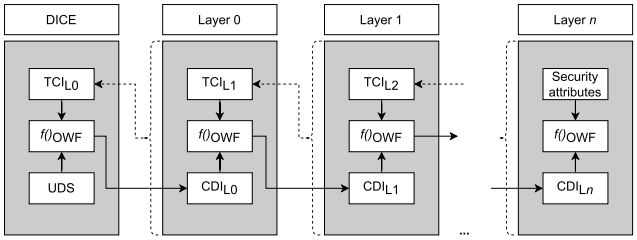


FIGURE 2. DICE layering architecture [13].

we performed to validate our architecture. Section VIII concludes the paper with final considerations on the proposed work and suggestions for possible future research.

II. BACKGROUND

A. TRUSTED COMPUTING

According to the TCG definition [11], an entity is considered *trusted* when it is assumed to behave as expected for the intended purpose. However, predictable behaviour does not inherently mean behaviour that is worthy of trust (e.g. we expect that a teacher will teach, and we expect that a cheater will deceive). To build trust in a computing platform, we need as foundation a trusted component whose misbehaviour cannot be detected during operation; this element is the so-called RoT [12]. From the RoT, it is possible to build a Trusted Computing Base (TCB), which includes all system components responsible for enforcing security policies. The most secure option is to implement a hardware RoT on a platform, through dedicated hardware modules to ensure high-security capabilities. The TCG states that three RoTs are necessary to build trust in a platform:

- the *RoT for Storage (RTS)* is responsible for securely storing sensitive data, protecting it from unauthorised access or tampering;
- the *RoT for Measurement (RTM)* is in charge of collecting a proof of the system's state (measure) and sending this information to the RTS;
- the *RoT for Reporting (RTR)* provides unforgeable and reliable evidence of the platform's state (as retrieved from the RTS), typically submitted to external entities, such as attestation services or remote verifiers.

A very common implementation of RTS and RTR is the TPM [11], proposed by the TCG. To guarantee a high-security level, it is typically implemented as a discrete chip, attached to the platform's motherboard so that it is available from the very first stage of the system boot. However, constrained devices often cannot accommodate a TPM module, so they need to rely on alternative hardware RoT solutions.

B. DEVICE IDENTIFIER COMPOSITION ENGINE (DICE)

The DICE [10] specification is a TCG proposal to create an RoT in hardware-constrained devices, where a TPM is impractical or infeasible. DICE enhances the security of such elements through the definition of secure device identities and attestation procedures, critical in the evolving IoT landscape. Its flexible architecture aligns with existing security standards, simplifying integration into current infrastructures.

DICE defines a layering architecture, introducing dependencies in the firmware and software layers of a device (Figure 2). This permits to adapt DICE to several classes of devices, from the simplest, including only one layer where is present a small firmware, to more complex ones, including several software and firmware layers (e.g. first stage bootloader, Unified Extensible Firmware Interface (UEFI)/Basic Input/Output System (BIOS), operating system). The platform must be equipped with a secret, the *Unique Device Secret (UDS)*, located in the most privileged firmware layer, called the DICE layer. The UDS is a unique secret for each platform, from which are derived the seed values, for the subsequent device layers. This seed is called *Compound Device Identifier (CDI)*. The purpose of the CDI is to derive the necessary keys of each layer in order to bind them to the device identity and the software layer integrity state. This value is computed for each device software layer, starting from the first mutable code (Layer 0), as follows:

$$CDI_{L0} = OWF(UDS, TCI_{L0})$$

This CDI associated with the first firmware layer (CDI_{L0}) is computed by combining through a One Way Function (OWF) [14] (e.g. SHA-256) the UDS and the measurement (typically as digest) of the associated code, called *TCB Component Identifier (TCI)*. So, the measurement of a layer refers to the cryptographic hash of its code. In this architecture, each layer has its own $CDI_{L(n)}$, calculated by combining the CDI of the previous layer ($CDI_{L(n-1)}$) with the measurement performed by the layer underneath ($TCI_{L(n)}$), as follows:

$$CDI_{L(n)} = OWF(CDI_{L(n-1)}, TCI_{L(n)})$$

This method of identity calculation permits the assessment of the integrity of the platform from the identity of the last layer. If all components are functioning correctly, the final CDI of a given application should give a deterministic value, indicating that the platform is trustworthy.

To effectively construct the DICE Layering Architecture, each layer must adhere to several essential hardware requirements [15]. First, each layer must implement some protection capabilities in order to be able to interact with other layers but not allow the disclosure of secrets. For this reason, each layer is responsible for its CDI and secrets, which must remain confidential from other layers. The UDS must be readable only by the DICE layer, ensuring it remains secret from other components; once the first CDI is derived, access to the UDS is permanently disabled through hardware mechanisms, ensuring it cannot be retrieved or reused until the next reset. All secrets within a layer must either be generated by that layer itself or provided by the previous layer. It is crucial that no layer implicitly trusts any layer above it, ensuring a robust security posture.

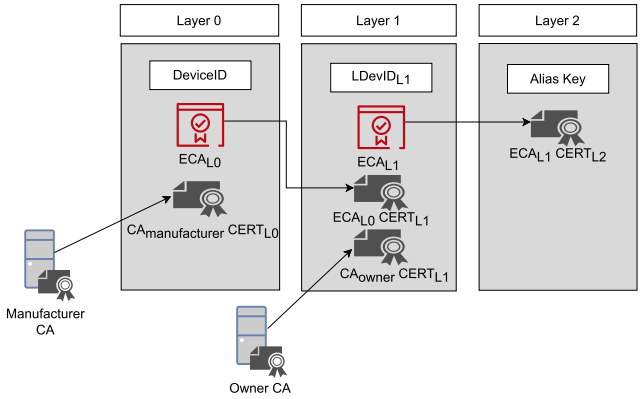


FIGURE 3. DICE layering architecture certificate hierarchy [13].

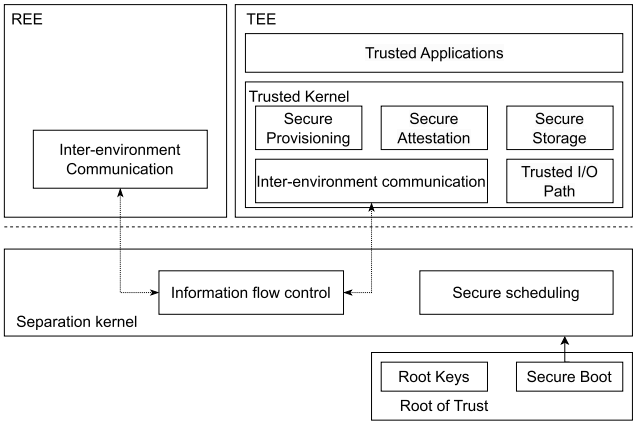


FIGURE 4. TEE building blocks [16].

The DICE specification [13] defines the derivation of cryptographic keys from the CDI of each layer, and how they should be certified (Figure 3). Two options are available: a public-key layer can be exported and associated with a Public-Key Certificate (PKC) issued by a trusted Certificate Authority (CA). Alternatively, it is possible to associate the key with a PKC generated inside the device, as each layer is able to issue certificates thanks to the definition of an Embedded Certificate Authority (ECA). In this way, starting from the manufacturer's CA, trustworthy credentials can be derived at each boot of the device, linking them directly with the device's integrity state. For the ECA scenario, there are two possible ways to issue a PKC for a specific layer:

1) the first possibility is a layer n to generate the keys and issue the related PKC. The keys and the PKC are passed by the layer n to the layer $n+1$ together with the $\text{CDI}_{\text{L}(n)}$.
2) the second possibility is a layer n to only generate the $\text{CDI}_{\text{L}(n+1)}$ and pass it to the correspondent layer $n+1$. Subsequently, the layer $n+1$ can generate its keys from its CDI, and request a PKC from the underneath layer n through a Certificate Signing Request (CSR).

In contrast, the procedure for issuing external certificates has to be performed with an *ad-hoc* operation by the manufacturer or the owner of the device. To effectively implement this chain, the device must have at least one certificate signed by the manufacturer, associated with the first public key generated.

C. TRUSTED EXECUTION ENVIRONMENT

A TEE [16] is a protected execution area, safeguarded from the rest of the system, called Rich Execution Environment (REE). Any data within the TEE cannot be accessed or altered by code outside this environment; therefore, only authorised code can manipulate the data inside the TEE [17]. A TEE guarantees the authenticity of executed code and the integrity of runtime states while securely storing code, data, and runtime states with confidentiality. It should enable RA to prove its trustworthiness to third parties. Additionally, TEEs are designed to resist both software and physical attacks on the system's main memory, preventing exploitation through backdoor vulnerabilities.

Four key principles define the TEE security requirements:

- *Data Separation* ensures that each protected area can access only its data, preventing unauthorised manipulation from other areas.
- *Sanitisation* prevents shared resources from being used to transfer information between protected areas, eliminating the risk of data leakage.
- *Information Flow Control* enforces that all communication between protected areas must be explicitly authorised.
- *Fault Isolation* guarantees that any security breach or system malfunction occurring within one protected area remains contained within that area, preventing the issue from propagating and compromising the integrity of other protected areas.

A TEE is composed of several key components (Figure 4) that work together to ensure its security and functionality:

- *Secure Boot* ensures that only verified code is run during system startup, preventing unauthorised software from running.
- *Secure Scheduling* ensures that TEE operations do not interfere with the performance of the main operating system, maintaining system efficiency.
- *Inter-environment Communication* facilitates secure interaction between the TEE and the rest of the system.
- *Secure Storage* maintains the confidentiality, integrity, and freshness of data, with access restricted to authorised entities.
- *Trusted I/O Path* secures communication between the TEE and system peripherals, protecting against attacks such as screen capture and overlay, and key logging.

Together, these components maintain the integrity of the TEE in a variety of environments.

The TEE concepts have been implemented for various architectures: ARM platforms with TrustZone [18], Intel

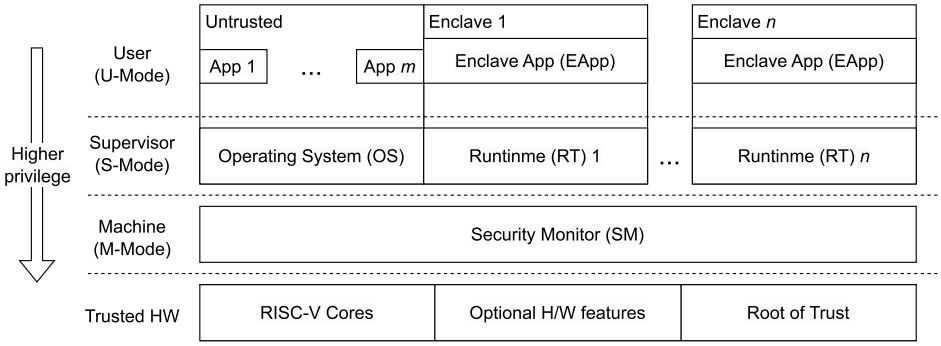


FIGURE 5. Keystone architecture [9].

platforms using the TDX [3], AMD platforms with Secure Encrypted Virtualisation (SEV) [19], IBM Z mainframes with IBM Secure Execution [20], and RISC-V platforms with Keystone [9].

TEEs are typically implemented in two possible ways. The first way is by creating a single ''secure world'' where all the secure applications are deployed. In this case, there is no isolation among secure applications, but only between REE and TEE. The second way is by introducing the *enclave* instance, which permits to obtain also isolation among secure applications. In this scenario, each application is deployed in an enclave, where it has its own trusted kernel instance, removing the possibility of directly communicating and sharing data with the REE and other enclaves.

D. THE KEYSTONE TEE FRAMEWORK

Keystone [9] is an open-source enclave-based TEE for RISC-V [21] processors, which exploits hardware capabilities for enforcing runtime protection and management. *Enclaves* are a special type of TEE implementation, designed to host sensitive portions of an application, to be executed being protected by the privileged code of the main Operating System (OS). Each enclave has an untrusted application responsible for managing its lifecycle. Keystone enclaves do not use a single OS because they are independent of each other and communicate with lower levels separately. Keystone leverages the Physical Memory Protection (PMP) RISC-V extension, which allows the division of the physical memory into regions and controls memory access permissions. It is based on three different privilege levels, to limit the capabilities of each layer.

Keystone has a layered architecture made up of several components that operate at different execution privileges. (Figure 5). At the lowest layer, there is the *trusted hardware*, which includes a RISC-V compliant CPU supporting the PMP security extension, a hardware RoT and, optionally, additional security features to implement cache partitioning or memory encryption. On top of the trusted hardware is the *SM*, which operates at the highest privilege level of the platform, Machine mode (M-mode). It is responsible for managing the lifecycle of enclave applications and modifying PMP registers to prevent unauthorised access to memory regions. Above the SM, the *OS* and *Runtime (RT)* layers operate in Supervisor mode (S-mode), lower privileged than M-mode. The OS handles untrusted applications, whereas the RT manages the enclave applications, overseeing their system calls, page table management for virtual memory of enclave applications, and interactions with the SM. At the highest layer, both *untrusted applications* and *enclave applications (Eapps)* run in User mode (U-mode), which is the least privileged execution mode. The SM offers interfaces for enclave management through Supervisor Binary Interfaces (SBIs). The SBI definition is part of the RISC-V specification [22] and permits switching from a less privileged mode (e.g. S-mode) to M-mode.

The life cycle of an enclave occurs through the following phases.

- *Creation*: an untrusted application requests the creation of an enclave from the host OS, which allocates a contiguous region of physical memory, the Enclave Private Memory (EPM), and initialises it with the enclave's page table, runtime, and Eapp. The host OS then invokes the SM's *create* SBI function, which protects the enclave's memory by adding two new PMP entries, whose state is communicated to all cores. Finally, the enclave is measured and validated by the SM.
- *Execution*: the untrusted application requests the OS to start the enclave execution by invoking the SM's *run* SBI function. The SM starts the enclave on an arbitrary core and dynamically changes the configuration of the PMP registers for that core, to prevent other system components from accessing the enclave's memory addresses and, at the same time, to prevent the enclave itself from accessing and manipulating regions of memory not assigned to it. When the enclave's execution time expires, the SM re-enters and restores the PMP permissions to allow OS execution.
- *Destruction*: the host OS can request the SM to destroy an enclave at any time, by invoking the SM's *destroy* SBI. When this occurs, the SM cleans the EPM associated with the enclave and frees the related PMP registers as well.

Keystone ensures enclave protection by memory isolation, preventing unauthorised access to enclave memory by the host OS or other enclave applications. Furthermore, a dedicated RT for enclave page management prevents controlled channel attacks, as the host OS has no control over the virtual-to-physical address mapping of the enclave. Keystone also ensures host OS protection from enclaves, as they lack access to external memory and cannot modify page tables or disrupt host operations. In particular, an enclave cannot be interrupted during its execution; for this reason, it is periodically rescheduled by the SM. A machine timer ensures that an enclave cannot monopolise CPU resources, preventing denial-of-service (DoS) attacks against the OS. For protecting the SM, Keystone employs the same PMP mechanism to enforce isolation, ensuring that neither the OS nor enclaves can tamper with it.

Despite all these security mechanisms, Keystone has some limitations. It heavily relies on PMP for enclave creation, so the number of enclaves that can be created on the platform is limited by the number of available PMP registers. Furthermore, its security depends on the absence of bugs in the SM, RT, and enclave applications, which should be formally verified to ensure overall TEE security.

III. RESEARCH MOTIVATION AND CONTRIBUTIONS

Securing IoT and embedded devices demands strong guarantees on device identity, secure boot procedures, and standardised attestation mechanisms to maintain trustworthiness throughout their lifecycle. However, these guarantees are often missing or not fully implemented in devices currently deployed in the field. Solutions such as ARM TrustZone [18] allow the force of the secure boot of the Secure World, containing trusted OS, firmware and trusted applications, without, however, allowing remote attestation of the components deployed in it, e.g. trusted applications developed by third parties. Intel Software Guard Extensions (SGX) [23] allows the implementation of remote attestation of enclave applications; however, it relies on the Intel proprietary Enhanced Privacy ID (EPID) protocol and on the use of the Intel Attestation Service, which is responsible for performing the group signature verification performed by the attesting enclave. Conversely, open-source initiatives, like Keystone, offer flexibility and transparency but inherently lack compliance with standardised identity provisioning and secure attestation protocols, essential for robust device integrity verification.

Keystone [9], an emerging open-source enclave-based TEE framework targeting RISC-V architectures, provides valuable capabilities for isolated execution but suffers from critical shortcomings:

- *Lack of standardized identity management*: Keystone's default identity management is not based on a hardware RoT, but on a device identifier written in plain text inside an include file of the boot Read Only Memory (ROM) code, thus providing security guarantees in the early stages of platform startup remains outside the scope of Keystone.
- *No secure boot process*: By default, Keystone does not verify the integrity of its first mutable component (i.e. the SM), responsible for enforcing the security guarantees of the TEE and loading the Rich OS, thus leaving the platform potentially vulnerable to unauthorised modifications of critical components at boot time.
- *Absence of standardized attestation layering*: Keystone does not natively support the TCG DICE layering architecture [13], which is increasingly recognised as pivotal for scalable, secure, and standardised attestation practices across software layers for IoT and embedded systems.

To overcome the aforementioned limitations, this paper introduces an open-source implementation of a DICE-enabled HRoT, integrated with the SM firmware of Keystone. Additionally, we propose a novel incorporation of the DICE layering architecture into the Keystone SM, specifically designed to meet the requirements of resource-limited IoT and embedded environments. It is important to note that the proposed implementation enables standardised attestation of launched enclaves during device operation, leveraging a DICE-compliant HRoT. In this sense, our approach provides a foundation for a form of continuous verification of the device's state, by regularly monitoring the enclaves running on the device during operation and attesting their integrity in a standardised format compliant with the TCG - DICE specification. However, this periodic attestation primarily captures the integrity state of the SM and enclave applications at load-time. True continuous measurement, involving dynamic, real-time detection and reporting of integrity violations at enclave runtime, would require further research and architectural extensions beyond the current scope of this work.

The primary contributions of this research include:

1) *Development of a DICE-Compliant Hardware RoT and its Integration into Keystone*: We designed and implemented a hardware RoT compliant with the TCG DICE specification, leveraging a silicon-based solution, a Physical Unclonable Function (PUF) to securely generate a unique, hardware-rooted identity for the device. This robust identity foundation significantly enhances resistance against device cloning and spoofing attacks.
2) *Secure Boot for Keystone's SM*: Our solution securely verifies the first mutable component (Keystone's SM), thereby preventing unauthorised or malicious firmware modifications at the earliest boot stage.
3) *Implementation of Comprehensive DICE Layering Architecture*: We fully implemented the DICE layering architecture within the Keystone framework, providing standardised load-time attestation of enclaves and laying the foundational framework necessary for

future extensions, such as runtime enclave attestation and trusted communication channels directly linked to identity keys derived according to the DICE specification.

4) *Detailed Security and Performance Evaluation*: A thorough security analysis and practical evaluation demonstrate our approach's feasibility and effectiveness, highlighting its suitability and robustness for resource-constrained embedded and IoT devices.

By achieving these contributions, our work not only enhances Keystone's capabilities significantly beyond its original implementation but also establishes a solid foundation upon which robust, standardised runtime attestation and trusted communication methods can be seamlessly integrated in the future. This effectively positions our work as an advance toward the establishment of a secure, standardised and practically feasible security ecosystem for IoT and embedded devices.

IV. RELATED WORK

In recent years, significant research efforts have been devoted to securing IoT and embedded systems, particularly through the adoption of hardware-rooted trust and robust attestation mechanisms. Several prominent frameworks and architectures have emerged, each addressing secure boot, remote attestation, and trusted execution in different ways.

The TCG's DICE has defined a comprehensive layering architecture to generate cryptographic identities from hardware-rooted secrets and software measurements, providing a scalable and standardised method for device attestation and identity management. Despite its clear advantages, there is limited published literature on integrating the DICE layering architecture within enclave-based TEEs, especially for RISC-V systems. Earlier RISC-V enclave frameworks, such as MIT's Sanctum [8], Komodo [7], and Penglai [24], have provided foundational secure boot and attestation capabilities through custom-designed mechanisms.

Sanctum [8], among the earliest RISC-V TEEs, implements a secure boot process to validate the SM, which subsequently provides remote attestation using device-specific secrets derived from hardware-level entropy, such as PUFs. However, Sanctum predates the DICE specification and relies on a relatively flat attestation model without leveraging DICE's hierarchical identity derivation.

The Keystone framework [9], an open-source enclave-based TEE derived from Sanctum, similarly adopted Sanctum's attestation methods, relying on single-layer device identity derivation rather than employing DICE's structured layering. Keystone utilises hardware mechanisms, specifically the PMP feature in RISC-V processors, to manage secure enclave execution. However, Keystone originally lacked a standardised identity management and secure boot procedure, leaving it potentially vulnerable to unauthorised modifications to critical components at boot time. Keystone's attestation mechanism typically involved generating a simple enclave attestation report signed with a static device-specific key, without the formal layered identity hierarchy prescribed by DICE.

MIT's Komodo [7] addressed enclave security through formal verification techniques, and implemented secure boot concepts, where the SM's integrity is verified before runtime. Nonetheless, Komodo's attestation approach remained minimalistic and proprietary, without adopting the DICE layering model or detailed device identity derivations. Penglai [24], a recent and scalable enclave-based TEE, similarly followed classical secure boot and attestation methods with device-specific keys, avoiding the complexity and interoperability advantages offered by the DICE specification.

Outside of the RISC-V landscape, Google's Project Oak [25] has demonstrated the viability of integrating a DICE-inspired recursive measurement strategy within TEEs, highlighting growing interest in structured trust layering. However, this application remains limited to specific proprietary Virtual Machine (VM)-based Intel and AMD TEEs (e.g. AMD SEV - Secure Nested Paging (SNP) [26], Intel TDX [3]), and no extensive peer-reviewed literature documents the deployment of DICE in RISC-V-based enclave TEEs.

The Open Profile for DICE [27] defines a simplified version of the TCG DICE specification, aiming for easier implementation while preserving the core principles of secure device identity derived from software measurements. Unlike the full TCG DICE, which requires hardware compatibility and establishes cryptographic operations involving a UDS, the Open Profile relaxes some constraints, reducing the dependency on secure hardware RoTs. Although it shares similar goals regarding device identity and attestation, its focus on lower hardware security requirements makes it less robust for high-assurance environments, like those using TEEs, which we consider in this work.

Microsoft's Robust Internet-of-Things (RIoT) [28] provides a foundational framework for trust in IoT systems by focusing on minimal hardware resources and allowing for secure device identification and attestation based on a device secret. However, RIoT does not integrate with TEE technologies, meaning it lacks the isolation and protection guarantees provided by TEEs. Additionally, RIoT primarily focuses on boot-time attestation and key derivation without extending protection to software launched at runtime. In contrast, our work aims to provide continuous attestation and identification of security-sensitive software running in enclaves during device runtime. Other architectures ([29] and [30]) leverage the DICE specification to establish trusted device identities and mutual attestation capabilities in heterogeneous IoT environments, yet these frameworks also do not specifically integrate DICE into enclave-based TEEs, thus limiting their applicability to high-assurance isolated environments.

OpenTitan [31] is an open-source initiative designed by lowRISC, in partnership with Google and other commercial and academic partners, to create a secure silicon ecosystem

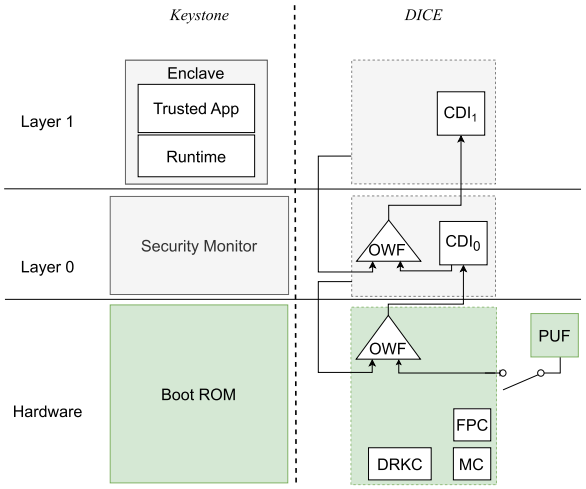


FIGURE 6. DICE specification layers mapping to the Keystone architecture.

capable of supporting numerous applications, including a discrete secure microcontroller providing RoT functionalities and an integrated SEE. While its versatility makes it an exciting project, OpenTitan faces problems similar to those of TPM-based solutions, especially regarding the impracticality of embedding trusted hardware components in low-cost IoT devices.

EPID [32] focuses on utilising anonymous attestation for encrypted communications, allowing devices to prove that they are genuine without revealing their specific identity. While it offers attestation guarantees, EPID is a closed-source system tied to Intel processors, which makes it unsuitable for non-Intel-based deployments.

The ARM Platform Security Architecture (PSA) [33] is a comprehensive framework developed by ARM to provide a standardised foundation for securing IoT and embedded devices. It leverages ARM TrustZone technology to create a system architecture with two separate execution environments, a secure world and a normal world. PSA defines guidelines for secure boot, attestation, key management, and overall device lifecycle security. By standardising these functions, PSA aims to facilitate the development of secure and interoperable devices. Despite its robustness, PSA remains platform-specific and does not directly translate to vendor-neutral architectures like RISC-V, limiting its interoperability.

The work proposed by Hoang et al. [34], describe a TEE architecture based on two different RISC-V cores, hardware-separated, one for the TEE execution, and the other for critical data and operations management, such as keys and secure boot. The authors propose an architecture for secure boot and TEE initialisation, based on a static key hierarchy.

Precisely, the very first stage of the boot process is performed by an isolated subsystem, which possesses its own bus, Random-Access Memory (RAM), boot ROM, and ROM. The root in the key hierarchy is posed in the device's manufacturer, which signs the Root Key certificate with its private key. The Root Key is a per-class key for devices, which is common for an entire series of products. The public part of the Root Key is stored in the ROM of the isolated environment, which is used during the boot process to verify the signature of the Zero Stage Boot Loader (ZSBL). The device is then provided with a Device Key, which has been signed offline with the Root Key, and this is a per-device key used to issue the per-program generated keys. In this case, the Root Key and the Device Key are stored in the isolated ROM of the second core, not generating them on-the-fly. In contrast, our solution aims to derive the DRK at each boot of the platform, leveraging a PUF to avoid the possibility of recovering the private part. In addition, in this work, the Device identity key is not linked with the code fingerprint of the platform. In our solution, having this link permits to attest the whole boot chain of the device, and verify the device is in a trustworthy state.

Given the surveyed landscape, our work advances the state of the art by introducing a DICE-compliant hardware RoT into Keystone, and explicitly integrating the standardised DICE layering architecture into the SM's firmware. To our knowledge, our approach represents the first comprehensive integration of the DICE layering model within a RISC-V enclave-based TEE framework. This integration provides Keystone with a DICE-enabled hardware RoT, secure boot capabilities and formally structured identity derivation, significantly enhancing security guarantees compared to existing enclave TEEs such as Sanctum, Komodo, and Penglai. Our approach enables a standardised and practical form of continuous verification of the device's integrity state, and lays a solid foundation for future extensions, including runtime enclave attestation capabilities, to detect and respond to integrity violations that can happen during operation, and the establishment of trusted communication channels between Eapps, rooted in DICE-derived identities. These contributions position our work as an advance for standardised secure enclave designs, bridging the gap between industry standards and academic innovations within the evolving IoT and embedded system security landscape.

V. SOLUTION DESIGN AND IMPLEMENTATION

Our solution aims to implement the DICE specification as HRoT in the Keystone framework. In addition, we also propose a DICE extension to support the secure boot process, allowing only trusted firmware to execute on the device.

To integrate DICE, the Keystone components have to be mapped on the DICE architecture, to identify all the layers involved and define the relationships. This mapping is shown in Figure 6. In particular, the DICE layer is implemented inside the boot ROM of the device. This component is implemented in hardware to achieve a high-security level for the trusted functionalities associated with the DICE layer. In addition, to avoid storing the UDS in the boot ROM, we chose to retrieve it through a *PUF* [35]. A PUF is implemented as a separate hardware component, which

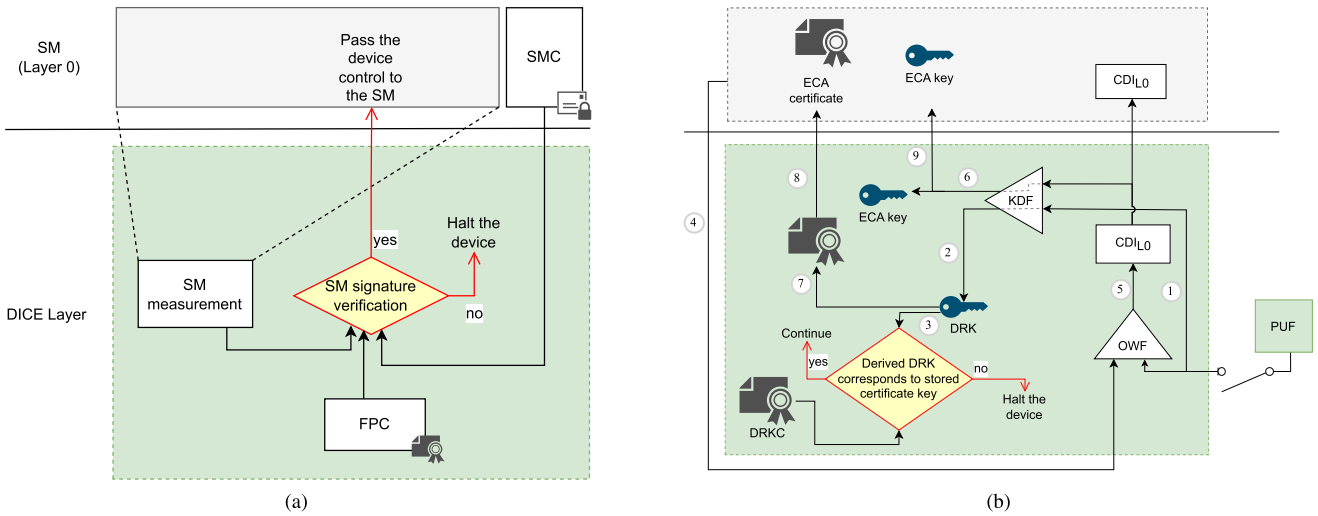


FIGURE 7. Operations performed during device boot: (a) Secure boot procedure. (b) Device Root Key (DRK) derivation and SM's ECA key derivation and certification.

leverages manufacturing randomness to create a unique "fingerprint" for the device. This ensures that the UDS cannot be physically reproduced and is unique to each device.

A. DICE LAYER

In addition to the UDS, which is never stored directly in memory but securely regenerated at each boot from the PUF module. The DICE layer is provided with the following data, generated during the manufacturer's initialisation phase:

- *Manufacturer Certificate (MC)*: the X.509 self-signed certificate of the manufacturer, serving as the root CA for all device certificates, optionally burned in the device boot ROM;
- *Device Root Key Certificate (DRKC)*: the X.509 certificate of the DRK signed by the manufacturer during the device initialisation protocol (see Section V-D);
- *Firmware Provider Certificate (FPC)*: the X.509 certificate of the SM provider (if different from the manufacturer), used to verify the signature of the SM during secure boot, burned at manufacturing time in the device boot ROM;
- *Security Monitor Certificate (SMC)*: the X.509 certificate of the SM, signed by the authorised firmware provider and validated during secure boot against the FPC.

The DRK is derived from the UDS at each boot and remains consistent as long as no unauthorised hardware modifications or identity metadata alteration generated during device initialisation (see Section V-D) occur. Importantly, neither the UDS nor the DRK are stored persistently on the device; they are securely re-derived at every reset using the PUF module, significantly reducing the risk of unauthorised extraction or duplication.

Our implementation of secure boot within Keystone (Figure 7a) enforces strict firmware verification procedures, effectively preventing the loading and execution of compromised SM firmware. This secure boot enforcement occurs during device boot, when the SM is loaded into main device memory by the boot ROM, ensuring the integrity of the SM through cryptographic verification using the FPC, persistently stored in the boot ROM. In order to do this, the DICE layer computes the SM measurement (TCI) and verifies it against the SM's signature, provided together with the system image of the device, and the FPC stored in the boot ROM. This secure boot procedure extends the DICE specification, preventing a compromised layer 0 (SM) from taking control of the device. Only upon successful verification of the SM signature does the DICE layer proceed with the derivation of the DRK, and the subsequent generation of keys and certificates for the SM.

After the SM signature verification, the DICE layer continues the device initialisation, which includes several critical steps (Figure 7b). First, the UDS value is securely retrieved from the PUF (1), leveraging mask and Helper Data created during the device initialisation process (see Section V-D) to regenerate the original unique secret. Next, the DRK is generated via a Key Derivation Function (KDF) seeded by the UDS (2), and its correctness is verified against the public key contained in the DRKC (3), provided to the device along with the system image. Subsequently, the SM measurement (TCI), computed during secure boot, is used to derive the CDI$_{L0}$ (4), by cryptographically combining the UDS and the SM TCI (5). The derived CDI$_{L0}$ is then used as a seed to generate the ECA key-pair via a KDF (6), and the corresponding X.509 certificate is issued and signed by the DRK (7). Then, the ECA key-pair and its certificate are securely passed to the SM (8) (9). Finally, all memory locations containing the UDS and DRK values are cleared to ensure no leakage of this critical sensitive data can happen to other software layers. Additionally, our

custom Keystone implementation explicitly blocks any direct software access to the PUF module's identifier functionality post-initialisation, making it impossible for an attacker to read or extract the original PUF-derived UDS. This hardware and firmware-level access restriction, combined with the robust secure boot procedure enforced by the immutable boot ROM, ensures comprehensive protection against any unauthorised attempts to retrieve or regenerate the device's unique cryptographic identity.

B. SECURITY MONITOR

Enclave creation is another important operation to take care of when integrating DICE in Keystone. This is because an enclave corresponds to a layer in the DICE architecture. For each enclave to be created, the associated CDI is computed starting from the CDI_{L0} and the TCI of the image of the enclave. In this case, our design differs from the DICE specification in the management of the keys of the enclaves. According to DICE, each layer should be responsible for its keys, so once it receives its CDI from the previous layer, it should generate and manage its keys. In this case, we decided to keep the creation and management of the enclave keys in the SM and open an interface to request their usage. This prevents an enclave from being migrated to a different device or from migrating its keys to a different device. Such a transfer would be an issue because each key-pair is tied to the hardware platform on which it is derived, being generated from the enclave's CDI and other platform data. The enclave's CDI, in fact, depends on the CDI of the SM, and so on all the way down to the HRoT. For this reason, moving the enclaves between platforms or migrating their identity keys would cause consistency issues due to this dependency chain. To avoid this, enclaves are designed without direct control over their key-pairs, requiring private key operations to go through the SM.

The enclave creation procedure, shown in Figure 8, begins with the measurement of the enclave image, generating the corresponding TCI (1). The $Enclave_{TCI}$ is then combined with the CDI_{L0} to generate through a OWF the $Enclave_{CDI}$ (2). This value is passed to a KDF to generate a key pair for the enclave (3). In this case, the created key is a Local Attestation Key (LAK) [36], which is the key that can be used to sign the attestation report generated by the SM [37], containing the integrity information (e.g. TCI) of the enclave. After the LAK creation, the SM ECA issues the corresponding certificate, which is then passed to the enclave together with the public part of the LAK (4), (5). This mechanism is the basis for enabling periodic attestation of the enclaves currently running on the device, supporting continuous verification of the device's TEE during operation.

Once the enclave starts, the SM exposes an interface to create other key-pairs, because the enclave does not have its CDI. In this way, the enclave can request the creation of a Local Device Identity (LDevID) key [38] and the corresponding certificate. This key is used to identify the

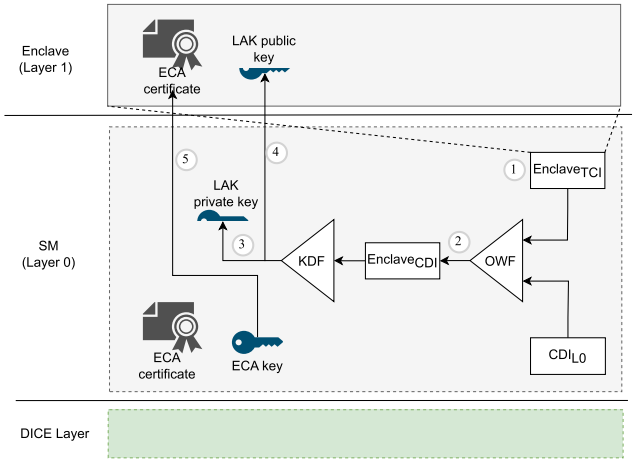


FIGURE 8. Operations for the enclave creation.

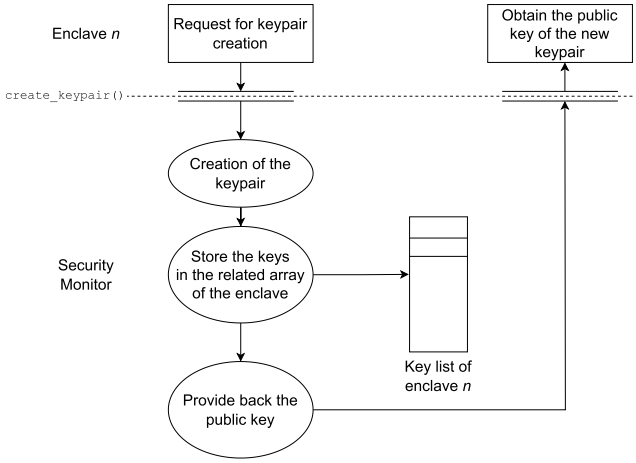


FIGURE 9. Creation of a new enclave key pair.

specific enclave as running on a specific device, because it is linked to the UDS through DICE. In this case, when a signature is required, the enclave provides the reference of the data to be signed on a shared buffer with the SM, the SM finds the related private key, performs the signature, and provides the result to the enclave. When it is necessary to verify the signature of an enclave, the complete certificate chain can be obtained from the SM. This chain includes the certificates for the manufacturer's key, the DRK, and the SM ECA, together with the enclave certificate. They allow a remote party to verify the validity of the enclave certificate. The SM certificate also contains an additional extension with the value of the SM TCI, which allows for verifying that the platform has booted correctly by comparing the extension with a reference value.

Critical functionalities implemented in hardware by the crypto-modules, such as the PUF used as the reference device identifier, are protected at the hardware access level by the SM by relying on the same PMP mechanism used in Keystone to protect unauthorized accesses to different memory regions

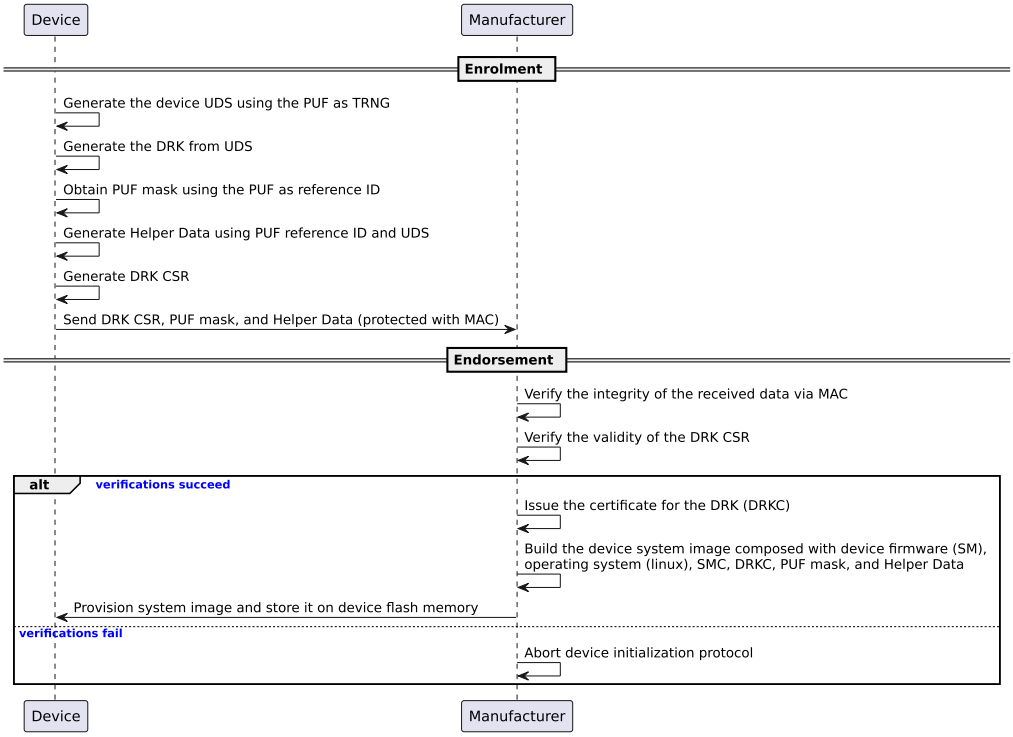


FIGURE 10. Device initialisation protocol.

(i.e. Rich OS, enclaves, SM). Specifically, the SM configures a PMP register at initialisation associated with the PUF identifier crypto-module, in a way to ensure that no enclave or user application can directly map or access the memory region used to communicate with the critical PUF functionality. Any attempt to directly access this protected memory region outside the authorised initialisation phase (at startup by the boot ROM) triggers hardware-level exceptions, preventing unauthorised reads or extraction of the PUF-derived UDS. In contrast, non-critical cryptographic functionalities implemented in hardware (e.g. AES, SHA2, SHA3, EdDSA, PUF used as True Random Number Generator (TRNG)) are exposed through controlled memory-mapped interfaces managed by the SM. This approach provides flexibility for application-level cryptographic operations while maintaining strict isolation and integrity for security-critical resources.

C. ENCLAVE KEY-PAIR CREATION

We developed a custom library for processing X.509 certificates according to DICE, as this is lacking in Keystone. We built it starting from the MBed TLS [39] library, which implements the necessary structures and methods for creating and parsing the X.509 certificates, such as Secure Hash Algorithm 2 (SHA-2) and Elliptic Curve Digital Signature Algorithm (ECDSA) for signature generation. The extension `DiceTCBInfo` [40] has been added to normal certificates to permit the insertion of the hash value of the related DICE layer. For instance, if the certificate is associated with the LAK of an enclave, `DiceTCBInfo` contains the enclave TCI.

As previously described, an enclave is not able to directly create and manage its LDevID. For this reason, we developed three new interfaces between the enclaves and the SM, implemented as SBI:

1) `create_keypair()` is the interface exposed by the SM for enclaves to request the creation of a new key-pair. This function returns the public part of the key-pair created, and the associated certificate issued by the SM ECA.
   This function requires a seed from the enclave, which is necessary to generate the key-pair and identify it in the enclave context. A schematic flow for the operation is represented in Figure 9.
2) `get_cert_chain()` interface permits to retrieve the whole device certificate chain, to enable the external verification of the LDevID certificate.
3) `do_crypto_op()` interface is used by enclaves when they need to perform crypto operations with the private key associated with their key-pair. This interface provides several operations requested via specific flags. For example, an enclave can retrieve the integrity report signed with its corresponding LAK, or it can request to sign some data with a specific key.

D. DEVICE INITIALISATION PROCESS

1) PUF IMPLEMENTATION AND ASSUMPTIONS

The security and reliability of our device initialisation depend on the PUF employed in the hardware-based RoT. This

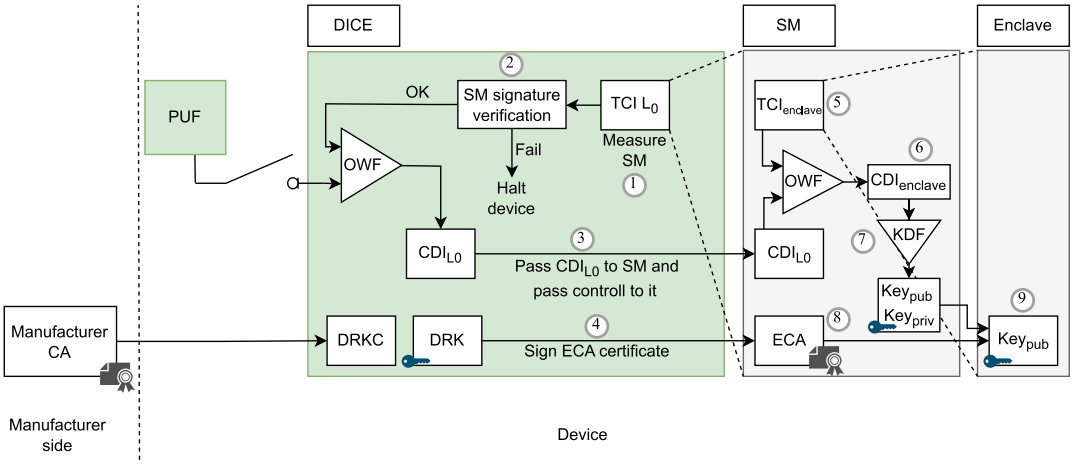


FIGURE 11. Procedure for secure boot and trustworthy enclave loading.

subsection provides details regarding the realisation of the PUF used in our design.

Our system incorporates a configurable PUF based on two Ring Oscillators (ROs) [41], whose uniqueness arises from variations intrinsic to the device manufacturing process. Each RO consists of several configurable logic stages, implemented on Field-Programmable Gate Array (FPGA) configurable logic blocks, enabling multiple oscillation frequencies depending on selected delay paths. The unique response, forming the device-specific identifier (PUF ID), is obtained by comparing the oscillation frequencies of two selected ROs under the same conditions.

During the device initialisation process, the unique PUF reference ID is generated by applying a predefined sequence of configuration bits, referred to as ''challenges'', each of which sets specific delay paths within the ROs, determining their oscillation frequencies. For each pair of ROs selected by a challenge, their frequencies are measured by independent counters until one of the counters reaches saturation. Bits from the slower RO's counter are then extracted, producing stable and reliable PUF identifier bits.

To enhance the reliability of the PUF identifier, the design includes a challenge-selection mechanism. This mechanism operates during the *enrolment* phase, where each available ''challenge'' is tested and characterised. Challenges resulting in less stable, unreliable responses (i.e. higher sensitivity to environmental or operational variations) are excluded. Thus, the identifier generated during device operation relies only on stable and robust challenges, significantly improving the consistency and reliability of the PUF ID.

Furthermore, to ensure stable and consistent regeneration of the device's UDS across device reboots, two important sets of information are generated and stored during the initial *enrolment* phase: the PUF *Mask* and *Helper Data*. In our design, the Helper Data represents the UDS XORed with the stable PUF reference ID, which is used as the secret symmetric encryption key for the UDS randomly generated during the initialisation process. The PUF Mask, on the other hand, is a bitstream which allows for selecting the most stable and reliable bit positions within the full PUF ID response. By masking out unstable or unreliable bits that could introduce inconsistencies, the PUF Mask significantly enhances the reliability of the PUF reference ID. Both Helper Data and PUF Mask constitute non-sensitive, publicly storable information that is maintained in the device's flash memory. These data are exclusively utilised by the boot ROM during device startup to reliably regenerate the original UDS established during initialisation. Possessing knowledge of the Helper Data and PUF Mask alone does not pose a security risk, as their use requires access to the PUF module's identifier functionality. Such access is explicitly disabled after device initialisation by our custom Keystone SM, effectively mitigating unauthorised utilisation and UDS regeneration by code executing in S-mode or U-mode.

Experimental validations confirm that the integrated RO-based PUF achieves high uniqueness (inter-device Hamming distances consistently around 48%), strong reliability (intra-device Hamming distances below 2%), and balanced outputs, ensuring unpredictability and robustness against environmental variations [41].

2) DEVICE INITIALISATION

For DICE to function effectively, each device requires an individual UDS to generate its corresponding DRK. Since the manufacturer cannot anticipate this unique device-specific key pair, secure initialisation is required at the first boot to provision the associated DRKC, as illustrated in Figure 10.

The initialisation process, conducted post-manufacturing, encompasses two distinct phases. The *enrolment phase*, performed on the device to be initialised, involves the autonomous generation of a unique UDS using the embedded PUF module, along with the corresponding Helper Data and Mask, essential for reliably regenerating the identical

UDS during next device startups. The subsequent *endorsement phase*, performed on the manufacturer's provisioning appliance, involves the validation of the device-generated data and the issuance of the certificate (DRKC) for the device's cryptographic identity (DRK) derived from the UDS. To ensure security through initialisation, the manufacturer installs a specially signed initialisation firmware on the device, temporarily stored, along with its signature, in flash memory. The immutable boot ROM enforces secure boot procedures to verify the integrity and authenticity of this firmware.

During the *enrolment* phase, the device first leverages the PUF module as a TRNG to produce a unique, unpredictable UDS, which seeds a cryptographic OWF to derive an ECDSA-based DRK using the Edwards curve 25519 (Ed25519) curve. Subsequently, the device generates a CSR for the DRK and securely transmit it to the manufacturer's provisioning appliance, alongside the PUF Mask and Helper Data. All sensitive data transmitted during the initialisation, including the DRK CSR, PUF Mask, and Helper Data, require secure protection of both integrity and confidentiality. Two distinct initialisation scenarios can be identified, which imply different levels of protection for the data generated during the enrolment phase and transmitted to the manufacturer's provisioning appliance. The first scenario, *local initialisation*, typically occurs within the manufacturer's secure environment, where the device communicates directly with the manufacturer's provisioning appliance through a physical point-to-point connection. Given this controlled and physically secure environment, the risk of external interception or unauthorised access is significantly mitigated. However, our design incorporates additional security measures, specifically integrity verification through a Message Authentication Code (MAC). The MAC ensures that the transmitted data remains authentic and unaltered, and may also serve operational purposes, such as authenticating firmware versions or particular configuration data within the initialisation firmware's settings. The MAC is generated using a key which is embedded in the bootrom used for manufacturing. Our current implementation and testing primarily address this local initialisation scenario, relying on direct physical connectivity and inherent environmental protections.

In the second scenario, *remote initialisation*, device initialisation might occur remotely, involving the transmission of sensitive data over potentially insecure networks, such as those accessible by clients or third parties. Although remote initialisation represents an interesting scenario for broader real-world deployments, our current design and implementation do not consider it, focusing specifically on secure initialisation performed locally at the manufacturer's premises.

In the subsequent *endorsement* phase, the manufacturer validates the received CSR and the associated data, ensuring their integrity and authenticity. Once validated, the manufacturer securely issues the certificate for the device's DRK (DRKC). This trusted certificate establishes the device's cryptographic identity within the ecosystem. The manufacturer then provisions the final device system image, which has to be recorded on the device's flash memory. This system image comprises the device firmware (the Keystone SM), an operating system image (Linux), firmware signature, the issued DRKC, and the previously generated PUF Mask and Helper Data. From subsequent boots onward, the device consistently regenerates the original UDS generated during the initialisation process and the certified DRK, provided no unauthorised modifications to the device hardware or the identity metadata (PUF Mask and Helper Data) occurred.

To explicitly mitigate the threat of unauthorised reinitialisation attempts or factory resets, our design integrates robust hardware- and firmware-enforced security measures. In particular, full PUF functionalities are restricted solely to immutable boot ROM code. Our custom firmware (Keystone SM) allows Eapps and REE applications to access only the TRNG functionality of the PUF module, preventing any access to the sensitive device identity regeneration functions to code running in S-mode and U-mode. Furthermore, the secure boot process, implemented by the immutable ROM, allows only firmware signed by the authorised firmware provider to run, thus effectively blocking, at the hardware level, malicious code running in M-mode to perform unauthorised initialisation attempts. Attackers attempting to replicate the manufacturer's initialisation protocol would have to load custom firmware to access the PUF identity functionalities, disabled in our SM, which would cause the secure boot verification to fail. Even in the hypothetical scenario of bypassing secure boot, practically infeasible, any newly generated UDS and derived DRK would lack an authentic manufacturer-issued certificate (DRKC), which would be immediately detectable during verification processes.

We clarify that the initialisation firmware, being crucial for the secure generation of the device identity, requires careful management and secure storage by the manufacturer or authorised third-party firmware provider. However, considering the potential case where an attacker gains access to the legitimate initialisation firmware, the following scenario emerges. The attacker might attempt to re-initialise the device, generating new Helper Data to establish a new device identity. Although technically possible, such re-initialisation would not pose a real threat since the new identity would lack an authentic manufacturer-issued certificate (DRKC). Without a DRKC, a remote verification process would immediately detect this unauthorised and uncertified identity. This assurance relies on the manufacturer's CA which, in the case of local initialisation at the manufacturer's premises, is not an online service and issues valid certificates exclusively during legitimate initialisation processes. An extreme scenario involving a simultaneous leakage of both the initialisation firmware and the CA's private key is theoretically possible but remains outside the scope of this work. Such an incident represents a catastrophic breach of the manufacturer's operational security and key management

practices, exceeding typical threat models and standard security assumptions for device initialisation processes.

3) DEVICE ZEROIZATION

Device *zeroisation* refers to the secure process of removing all sensitive user-specific information and cryptographic keys from a device, effectively returning it to a factory-default state. According to the TCG, zeroisation typically involves deleting non-factory-installed configurations and keys, including locally generated identities (LDevID) and other operational data, but normally preserving the device hardware identity installed during manufacturing [42]. The primary goal of zeroisation is to prevent unauthorised access to sensitive information by securely clearing data when devices change ownership or are repurposed.

Unlike typical zeroisation approaches that preserve the initial device identity key, our design offers an enhanced form of zeroisation capability, wherein even the hardware-rooted cryptographic identity can be regenerated. In particular, our design supports the generation of a new UDS and corresponding PUF-based Helper Data, without needing any hardware change, only requiring a complete re-initialisation process to establish a new certified device identity. This is enabled by a peculiarity of our design, where the PUF reference ID is never used directly as a device identifier, but is used as a symmetric key to encrypt the UDS generated during the initialisation process. Obviously, the revocation of the DRKC associated with the old device identity ($DRKC_{old}$), is also required by the manufacturer CA.

To ensure the legitimacy of this enhanced zeroisation process, our design imposes strict manufacturer-level controls, already discussed for the general initialisation process. In particular, reinitialisation is permissible exclusively via manufacturer-authorised firmware, validated by secure boot procedures enforced by immutable boot ROM. Even if an attacker gained access to leaked initialisation firmware, any newly created device identity would lack the manufacturer-issued certificate, immediately identifying the device as untrusted during remote verification. Thus, the effectiveness and security of the reinitialisation process rely fundamentally upon the robust operational security of the manufacturer's provisioning infrastructure and CA, which ensures trustworthiness and authenticity of new device identities. By enforcing these protection measures, our design provides flexibility beyond traditional zeroisation procedures, allowing complete removal and regeneration of the device identity without compromising security.

VI. SECURITY ANALYSIS

This section provides a comprehensive security analysis of our proposed integration of the TCG DICE architecture into Keystone, highlighting enhancements compared to the original Keystone implementation and other prominent RISC-V and non-RISC-V enclave-based TEEs.

The core security enhancement provided by our design consists of implementing a standardised, layered identity management mechanism, accurately reflecting the trustworthiness state of individual software components, coupled with a robust, secure boot procedure. By embedding a robust hardware-rooted identity, from which the identity of the software components constituting the TEE is derived, and a cryptographically verifiable boot sequence starting from the boot ROM, our solution significantly mitigates threats related to unauthorised modifications of the critical initial firmware and software components.

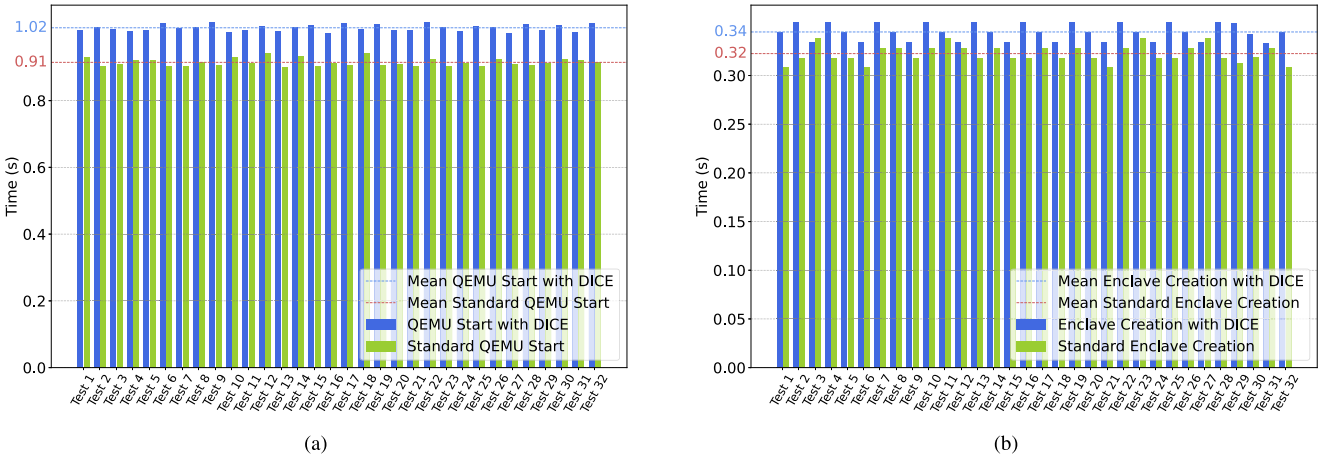
Table 1 provides a detailed comparative analysis of our proposed implementation against original Keystone [9], Sanctum [8], Komodo [7], Penglai [24], Intel SGX [23], and Intel TDX [3] in terms of security features, potential risks, hardware dependencies, resource overhead, and performance overhead. Although Intel SGX and TDX differ significantly from our Keystone-based work in terms of architecture (proprietary Intel x86 versus open-source RISC-V), they are included here as widely recognised industry-standard benchmarks. Their inclusion allows highlighting key advantages and differences of our RISC-V based DICE-integrated solution against established commercial offerings, providing additional insights into our approach's positioning. The comparative analysis presented in table 1 clearly demonstrates that our solution combines the benefits of open-source hardware compatibility, minimal performance overhead, and high interoperability with standardised, robust security mechanisms. Unlike original Keystone and Penglai, which rely on static identities vulnerable to extraction and cloning, our DICE-based hierarchical identity management provides strong protection against spoofing attacks through hardware-derived, cryptographically secure layered identities. Moreover, compared to proprietary solutions such as Intel SGX and TDX, our implementation offers significantly improved interoperability by adhering strictly to the TCG DICE standard, enabling seamless integration into diverse security ecosystems.

A. SECURITY RISKS MITIGATED

Our proposed design (Figure 11) addresses several security risks inherent in the original Keystone implementation. Specifically, leveraging a hardware-based PUF to derive the UDS, our design substantially reduces the risk of identity spoofing and cloning attacks compared to static device identity solutions. Furthermore, integrating secure boot verification of Keystone's SM mitigates the risk of unauthorised or malicious firmware execution at boot time. This permits the enforcement of the correct boot of the TEE. To obtain this behaviour, we assume that the boot ROM of the device is a RoT for the device, certified by the manufacturer. This means that the integrity of this component is never questioned and, in case of compromise, it would not be detected. With this assumption, we can build trust during the boot process, up to the SM, which will not start if the signature verification by the DICE layer fails. This verification is performed at boot time, hence detection of run-time attacks against the SM is out of scope. Once the

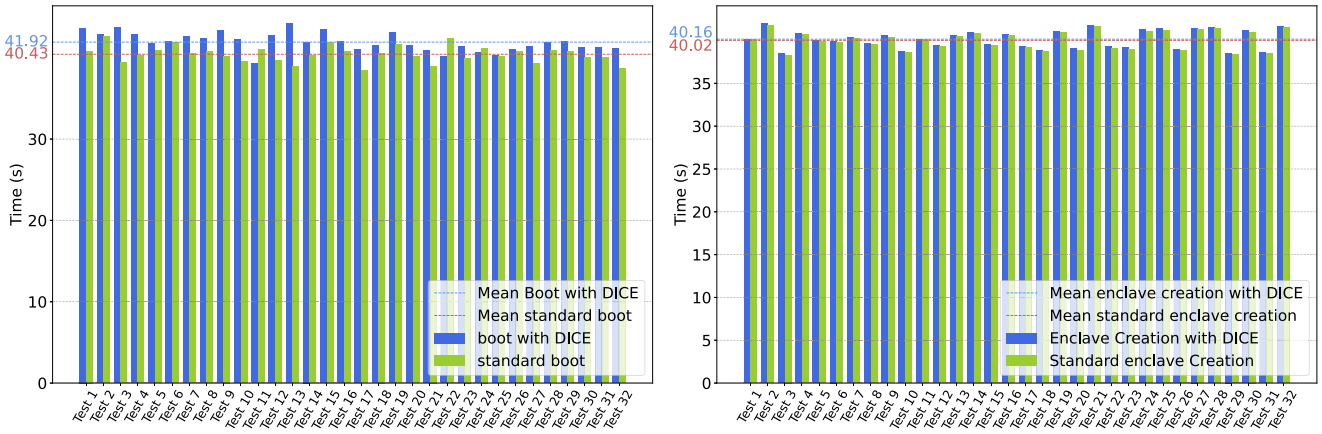
TABLE 1. Security and feature comparison of TEE implementations.

Feature / Aspect	Original Keystone [9]	Sanctum [8]	Komodo [7]	Penglai [24]	Intel TDX [3]	T.-T Hoang *et al.* [34]	Our Solution
Secure Boot	✗ No	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes
Identity Derivation	Static, single-layer	Static, single-layer	Ephemeral, per-enclave transient AK	Static, single-layer	Ephemeral (per-VM derived from CPU keys)	Static, hierarchical	**Hierarchical, DICE-compliant layered**
Hardware Root-of-Trust	Boot ROM + static device key	Boot ROM + PUF-derived device key	ARM TrustZone Secure Boot	Boot ROM + static device key	CPU integrated	Boot ROM + isolated core + static device key	**Boot ROM + PUF-derived UDS + DRK Certificate (DICE compliant)**
Load-time Attestation	⚠ Proprietary	⚠ Proprietary	⚠ Proprietary	⚠ Proprietary	⚠ Proprietary Intel-specific	⚠ Proprietary	✓ **Standardized (TCG-DICE identity chain)**
Runtime Attestation	✗ None	✗ None	✗ None	✗ None	✗ None	✗ None	⟳ **Future Extension**
Spoofing Resistance	⚠ Weak (static identity easily cloned if extracted)	✓ Strong (PUF-derived unique identity)	✓ Strong (ephemeral identities + TrustZone-enforced isolation)	⚠ Weak (static identity easily cloned if extracted)	✓ Strong (hardware-protected ephemeral identities)	⚠ Weak (static identity easily cloned if extracted)	✓ **Strong (DICE-compliant layered identity, PUF-derived)**
Resource Overhead	✓ Minimal (PMP-based, lightweight SM)	⚠ Moderate (cache partitioning)	⚠ High (formal verification overhead)	⚠ Moderate (extended PMP logic)	⚠ High (memory encryption, VM isolation overhead)	⚠ High (additional RISC-V core and per-core memory)	✓ **Minimal (standard DICE, lightweight cryptography)**
Implementation Complexity	✓ Low (basic PMP usage, minimal TCB)	⚠ Moderate (custom hardware extensions, cache partitioning)	⚠ High (formal verification, TrustZone-specific logic)	⚠ Moderate (extended PMP logic)	⚠ High (VM management, memory encryption, protection mechanisms)	⚠ High (dedicated core for secure boot, dedicate memory for secure boot external TRNG)	⚠ Moderate (DICE integration, cryptographic key layering)
Standard Compliance (RA)	✗ None (custom attestation)	✗ None (custom attestation)	✗ None (custom attestation)	✗ None (custom attestation)	✗ None (fully proprietary)	✗ None (custom attestation)	✓ **Full (TCG-DICE compliant)**
Interoperability	⚠ Low	⚠ Low	⚠ Low	⚠ Low	⚠ Moderate (standard certificates, proprietary quote format)	✓ High (standard certificates)	✓ **High (standard certificates)**
Hardware-Friendliness	✓ RISC-V PMP	⚠ Custom RISC-V core (Rocket), PUF	⚠ ARM TrustZone	✓ RISC-V PMP	⚠ Intel CPUs	✓ RISC-V PMP	✓ **RISC-V PMP, PUF**
Formal Verification	✗ No (empirical testing)	✓ Yes (formal proofs of enclave isolation)	✓ Yes (Secure Monitor and world-switch code)	✗ No (empirical testing)	⚠ Partial (some core components formally verified, others empirically tested)	✗ No (empirical testing)	✗ No (empirical testing)
Open Source Availability	✓ Yes (fully open source)	✓ Yes (fully open source)	⚠ Partially (software open-source, hardware proprietary)	✓ Yes (fully open source)	⚠ Partially (SDK open source, hardware proprietary)	✓ Yes (fully open source)	✓ **Yes (fully open source)**
Performance Overhead	✓ Minimal (basic PMP usage)	⚠ Moderate (cache management)	⚠ Moderate (lightweight enclave switching operations)	⚠ Moderate (extended PMP logic)	⚠ High (memory encryption, VM isolation overhead)	⚠ Moderate (double core boot)	✓ **Minimal (4% boot overhead, 0.3% enclave overhead compared to basic Keystone)**



(a) (b)

FIGURE 12. Performance of Keystone with the DICE integration in the virtualised environment: (a) Time spent during DICE initialisation and secure boot compared with the original Keystone implementation in the Quick Emulator (QEMU) setup. (b) Time spent during the enclave creation in the case of DICE operations compared with the original Keystone implementation in the QEMU setup.



(a) (b)

FIGURE 13. Performance of Keystone with the DICE integration on the Genesys2 physical board: (a) Time spent during DICE initialisation and secure boot compared with the original Keystone implementation in the Genesys2 board. (b) Time spent during the enclave creation in the case of DICE operations compared with the original Keystone implementation in the Genesys2 board.

boot procedure terminates correctly, the integrity of the SM is protected at the hardware level by the PMP mechanism, which allows intercepting and denying any illegal access to its memory area.

B. STANDARD COMPLIANCE AND INTEROPERABILITY

Adhering to the TCG DICE specification ensures our solution provides standardised, verifiable attestation mechanisms, significantly enhancing interoperability across diverse IoT ecosystems. This compliance contrasts markedly with the proprietary attestation methods employed by existing frameworks, positioning our work as a significant step forward for standardised device identity and integrity verification.

The trust in the device manufacturer, which endorses the platform trustworthiness through the DRKC, makes it possible to implicitly verify the integrity of each enclave through its identity keys. To perform this verification, a third party must possess the certificate of the enclave with which it wants to communicate. When receiving the enclave certificate, the third party can verify if the enclave was loaded correctly and determine its trustworthiness. Also, this standardised attestation enables periodic integrity verification of the device enclaves, thus providing effective verification that the running enclaves are healthy and no "rogue" enclaves have been launched on the device. However, it is important to emphasise that this verification reflects only the integrity state of the enclaves at load-time and the current executing enclaves during periodic attestation intervals. Continuous measurement, involving runtime tracking and reporting of enclave state changes, is not supported in this work and represents a crucial area for future extensions.

C. RESOURCE OVERHEAD AND HARDWARE-FRIENDLINESS

Our implementation introduces moderate resource overhead associated with cryptographic operations required by the

DICE-compliant layered key derivation and secure boot procedures (see Section VII). However, this overhead remains suitable for resource-constrained IoT and embedded devices, carefully balancing security enhancements with performance constraints. Utilising standard RISC-V PMP and PUF hardware components, our approach remains hardware-friendly and broadly compatible with existing hardware solutions. This contrasts with implementations such as Intel SGX [23] and TDX [3], which involve extensive hardware modifications and resource-intensive mechanisms like memory encryption and paging.

D. FORMAL VERIFICATION AND ASSURANCE LEVEL

While our current implementation primarily relies on empirical testing, the robust security guarantees provided by our solution significantly mitigate security risks even in the absence of formal proofs. Specifically, the standardised and verifiable cryptographic chaining provided by the TCG-DICE protocol, combined with secure PUF-based hardware identities that remain unknown even to the manufacturer due to our proposed protocol, provides strong assurances. Moreover, our DICE-enabled hardware Root-of-Trust, securely embedded in boot ROM, along with isolation guarantees enforced by the PMP-based TEE (Keystone), collectively establish a trustworthy foundation robust against various attack vectors. Future developments could include formal verification of critical components, similar to Sanctum [8] and Komodo [7], to further enhance assurance levels.

E. LIMITATIONS AND FUTURE EXTENSIONS

The current scope of our work focuses on secure boot and standardised DICE-compliant load-time attestation for enclaves. As a result, runtime compromise detection through continuous measurement (i.e. continuously detecting and cryptographically reporting any state changes or integrity violations occurring within enclave applications at runtime) remains beyond the current scope. Future research will explore the integration of such continuous measurement capabilities to provide dynamic, real-time security guarantees. Building upon the strong foundational identity management and secure initialisation established by this work, these extensions could significantly enhance security in highly demanding and dynamic IoT deployments.

VII. TEST AND EVALUATION

Our design of DICE in Keystone has been implemented and tested on both a virtualised environment and a physical board. The tests conducted evaluate how much the DICE integration[1] impacts the Keystone performance.

A. VIRTUAL TESTBED

All the features introduced from the booting phase to the SM have been successfully tested using the QEMU. All the tests were conducted on a single physical machine that supported the configuration and installation of Keystone. In particular, the platform hosting QEMU was an Intel Next Unit of Computing (NUC) with the following configuration:

- CPU Intel i5-5300 processor clocked at 2.30 GHz, four cores, eight threads;
- RAM 16 GB;
- OS Ubuntu 22.04 Long-Term Support (LTS) 64 bit.

B. VIRTUAL TESTS

The tests in the virtualised environment aimed to verify the performance of different operations.

We measure the time up to the start of the SM.

Figure 12a compares the performance of plain Keystone versus our implementation with secure boot and the activation of the DICE layer. The latter has a 12% overhead with respect to the plain Keystone boot. This overhead is due to the need for generating an X.509 certificate and verifying the manufacturer's signature over the SM. It should be noted that the measurement of the SM is also performed in plain Keystone, and that result is reused in our solution, with no need for an extra operation.

Figure 12b shows the performance for enclave creation. It can be seen that the performance is not much affected by the introduction of the new operations. These are mainly related to the creation of the X.509 certificate and of the LAK for the enclave.

C. PHYSICAL TESTBED

The tests were also performed on a physical board with a modified version of Keystone. The board is a Genesys2 Kintex-7 FPGA (XC7K325T-2FFG900C) [43] with 1 GB of Double Data Rate Type 3 (DDR3) RAM, 256 Mbit Quad serial Flash. It has a CVA6 RISC-V core, which is a 6-stage, single-issue, in-order CPU which implements the 64 bit RISC-V ISA. This board has a completely programmable boot process. In this way, it was possible to implement in the FPGA a PUF module [41] to generate the UDS for DICE. Additionally, the FPGA offered various hardware-based cryptographic [44] operations, and they were used in our implementation to optimise performance.

D. PHYSICAL TESTS

The boot ROM is synthesised and flashed on the board, and the manufacturer correctly generates a certificate for the board DRK. An Secure Digital (SD) card with three partitions is used for the test. The first partition contains the modified SM code and a minimal Linux image that will be launched after the SM. The second partition has a minimal filesystem for the Linux partition. The last one is used for the additional data necessary for DICE: the signature of the SM, the DRK certificate, and the manufacturer and firmware provider certificates.

Also in this case, the first test measures the impact of implementing the DICE specification on the platform boot. As shown in Figure 13a, the additional overhead to the boot

[1] https://github.com/torsec/keystone-dice

procedure is negligible. The boot of plain Keystone takes 40.43 s on average, while the boot with DICE support takes 41.92 s. This results in an overhead of 1.5 s or 4%, which is acceptable for such an important security feature. This overhead is due primarily to signature verification during the secure boot process.

Figure 13b reports the overhead for enclave creation. In this process, new keys (LAK) are generated from the TCI of the enclave code. The measured overhead can be considered negligible, being in the order of milliseconds. In particular, the average enclave creation time for plain Keystone is 40.07 s, while our DICE integration ticks at 40.21 s, or a 0.3% difference.

E. GENERAL EVALUATION

In conclusion, the tests demonstrate that the DICE integration and the secure boot extension introduced in Keystone do not significantly impact the performance. In particular, the overhead is mostly due to the generation of keys and certificates, plus signature verification, during the boot and the enclave creation.

VIII. CONCLUSION

The adoption of IoT devices is rapidly growing, especially in critical scenarios and everyday use cases. For this reason, providing them with strong security properties is becoming mandatory. The design we propose aims to obtain all the necessary security features to enforce and verify a platform's integrity and its correct operation. We chose the Keystone framework, based on security extensions of the RISC-V instruction set architecture. Despite the high security introduced by the TEE, Keystone lacks an HRoT for strong integrity and identity of devices and applications. To overcome this lack, we integrated the DICE specification as an HRoT, permitting to build a secure environment that provides strong integrity for critical applications. We performed several tests to verify the correctness and performance of our implementation. Our modifications introduced additional overhead compared to the original Keystone implementation; however, the overall execution time remains within acceptable limits. The small overhead introduced is due to a proper implementation of the DICE specification, which requires performing signature verification and measuring components during the device boot process, which are generally time-consuming operations.

The goal of this work has been achieved by introducing the DICE specification as HRoT in the Keystone TEE framework. Our implementation provides standardised attestation capabilities of enclave applications during device operation, thereby enabling continuous verification of the enclave running on the device through periodic integrity checks aligned with the DICE specification. However, this approach primarily captures enclave integrity at load-time and at defined intervals during device operation. Continuous measurement of enclave integrity states, involving real-time monitoring, detection, and reporting of integrity violations during runtime, represents an important future extension. Thus, future work includes extending our design toward true runtime attestation by implementing dynamic continuous measurement of running enclaves. Additionally, future work includes implementing implicit enclave integrity verification based on the key generated from DICE, allowing for a drastic reduction in the remote attestation overhead caused by the measurements collection and verification. These advancements could significantly enhance the security of IoT communications, enabling highly robust trusted channels suitable for dynamic and security-critical deployments.

REFERENCES

[1] J. M. Kizza, "Internet of Things (IoT): Growth, challenges, and security," in *Guide to Computer Network Security*. Cham, Switzerland: Springer, Jan. 2024, pp. 557–573, doi: 10.1007/978-3-031-47549-8_25.

[2] Statista. (2022). *Number of Internet of Things (IoT) Connections Worldwide From 2022 to 2023, With Forecasts From 2024 to 2033*. [Online]. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[3] P.-C. Cheng, W. Ozga, E. Valdez, S. Ahmed, Z. Gu, H. Jamjoom, H. Franke, and J. Bottomley, "Intel TDX demystified: A top-down approach," *ACM Comput. Surv.*, vol. 56, no. 9, pp. 1–33, Apr. 2024, doi: 10.1145/3652597.

[4] Qualcomm Technologies, Inc. *Guard Your Data With the Qualcomm Snapdragon Mobile Platform*. Accessed: Jul. 1, 2025. [Online]. Available: https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/guardyourdatawiththequalcommsnapdragonmobileplatform2.pdf

[5] Huawei. *Confidential Computing TrustZone Kit Feature Guide*. Accessed: Jul. 1, 2025. [Online]. Available: https://support.huawei.com/enterprise/en/doc/EDOC1100304592/40938d86/confidential-computing-trustzone-kit

[6] Linaro. *OP-TEE*. Accessed: Jul. 1, 2025. [Online]. Available: https://www.trustedfirmware.org/projects/op-tee/

[7] A. Ferraiuolo, A. Baumann, C. Hawblitzel, and B. Parno, "Komodo: Using verification to disentangle secure-enclave hardware from software," in *Proc. 26th Symp. Operating Syst. Princ.*, Shanghai, China, Oct. 2017, pp. 287–305, doi: 10.1145/3132747.3132782.

[8] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *Proc. 25th USENIX Secur. Symp.*, Austin, TX, USA, 2016, pp. 857–874. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan

[9] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: An open framework for architecting trusted execution environments," in *Proc. 15th Eur. Conf. Comput. Syst.*, Heraklion, Greece, Apr. 2020, pp. 1–16, doi: 10.1145/3342195.3387532.

[10] TCG. (Mar. 2024). *Hardware Requirements for a Device Identifier Composition Engine*. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/Hardware-Requirements-for-a-Device-Identifier-Composition-Engine-Version-1.0-Revision-0.91pub.pdf

[11] TCG. (Jan. 2024). *Trusted Platform Module Library Part 1: Architecture*. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/TPM-2.0-1.83-Part-1-Architecture.pdf

[12] TCG. *TCG Glossary—Version 1.1—Revision 1.00*. Accessed: Jul. 1, 2025. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/TCG-Glossary-V1.1-Rev-1.0.pdf

[13] TCG. (Jul. 2020). *DICE Layering Architecture*. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/DICE-Layering-Architecture-r19pub.pdf

[14] S. Chhabra, V. Dhanwani, V. K. Dhaka, and K. Lata, "Design and analysis of secure one-way functions for the protection of symmetric key cryptosystems," in *Proc. 24th Int. Symp. VLSI Design Test (VDAT)*, Jul. 2020, pp. 1–6, doi: 10.1109/VDAT50263.2020.9190432.

[15] TCG. *DICE Hardware Requirements*. Accessed: Jul. 1, 2025. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/Hardware-Requirements-for-Device-Identifier-Composition-Engine-r78For-Publication.pdf

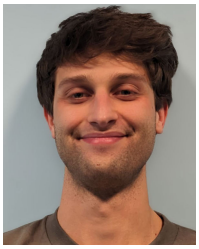
[16] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Helsinki, Finland, Aug. 2015, pp. 57–64, doi: 10.1109/TRUSTCOM.2015.357.
[17] M. Sabt, M. Achemlal, and A. Bouabdallah, "The dual-execution-environment approach: Analysis and comparative evaluation," in *Proc. 30th IFIP TC 11 Int. Conf.*, Hamburg, Germany, 2015, pp. 557–570, doi: 10.1007/978-3-319-18467-8_37.
[18] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, and S. Martin, "TrustZone explained: Architectural features and use cases," in *Proc. IEEE 2nd Int. Conf. Collaboration Internet Comput. (CIC)*, Pittsburgh, PA, USA, Nov. 2016, pp. 445–451, doi: 10.1109/CIC.2016.065.
[19] D. Kaplan, J. Powell, and T. Woller. (Oct. 2021). *AMD Memory Encryption*. [Online]. Available: https://www.amd.com/system/files/TechDocs/memory-encryption-white-paper.pdf
[20] IBM corporation. (Nov. 2022). *Introducing IBM Secure Execution for Linux 1.3.0*. [Online]. Available: https://www.ibm.com/docs/en/linuxonibm/pdf/l130se03.pdf
[21] A. Raveendran, V. B. Patil, D. Selvakumar, and V. Desalphine, "A RISC-V instruction set processor-micro-architecture design and analysis," in *Proc. Int. Conf. VLSI Syst., Architectures, Technol. Appl. (VLSI-SATA)*, Bengaluru, India, Jan. 2016, pp. 1–7, doi: 10.1109/VLSI-SATA.2016.7593047.
[22] RISC-V Platform Specification Task Group. *RISC-V Supervisor Binary Interface Specification*. Accessed: Jul. 1, 2025. [Online]. Available: https://www.scs.stanford.edu/
[23] V. Costan and S. Devadas. (Jan. 2016). *Intel SGX Explained*. [Online]. Available: https://eprint.iacr.org/2016/086
[24] E. Feng, X. Lu, D. Du, B. Yang, X. Jiang, W. Xia, M. Chen, W. Li, C. Lu, Y. Xia, B. Zang, and H. Chen, "Scalable memory protection in the PENGLAI enclave," in *Proc. 15th USENIX Symp. Operating Syst. Design Implement.*, 2021, pp. 275–294. [Online]. Available: https://www.usenix.org/conference/osdi21/presentation/feng
[25] *Project Oak*. Accessed: Jul. 1, 2025. [Online]. Available: https://github.com/project-oak/oak
[26] M. Misono, D. Stavrakakis, N. Santos, and P. Bhatotia, "Confidential VMs explained: An empirical analysis of AMD SEV-SNP and Intel TDX," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 8, no. 3, pp. 1–42, Dec. 2024, doi: 10.1145/3700418.
[27] Google. *Open Profile for Dice*. Accessed: Jul. 1, 2025. [Online]. Available: https://pigweed.googlesource.com/open-dice/
[28] P. England, A. Marochko, D. Mattoon, R. Spiger, S. Thom, and D. Wooten, "RIoT—A foundation for Trust in the Internet of Things," Microsoft Res., Tech. Rep. MSR-TR-2016-18, 2016. [Online]. Available: https://www.microsoft.com/en-us/research/publication/riot-a-foundation-for-trust-in-the-internet-of-things/
[29] E. Bravi, S. Sisinni, and A. Lioy, "Exploiting the DICE specification to ensure strong identity and integrity of IoT devices," in *Proc. 8th Int. Conf. Smart Sustain. Technol. (SpliTech)*, Jun. 2023, pp. 1–6, doi: 10.23919/SpliTech58164.2023.10193517.
[30] S. Sisinni, D. Gratiela Berbecaru, V. Donnini, and A. Lioy, "MATCH-IN: Mutual attestation for trusted collaboration in heterogeneous IoT networks," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Paris, France, Jun. 2024, pp. 1–6, doi: 10.1109/ISCC61673.2024.10733616.
[31] *The OpenTitan Project*. Accessed: Jul. 1, 2025. [Online]. Available: https://opentitan.org/
[32] E. Brickell and J. Li, "Enhanced privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities," *IEEE Trans. Dependable Secure Comput.*, vol. 9, no. 3, pp. 345–360, May 2012, doi: 10.1109/TDSC.2011.63.
[33] ARM Developer. (2019). *Platform Security Architecture*. [Online]. Available: https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/PSA/DEN0083PSATBSA-M1.0-bet2.pdf
[34] T.-T. Hoang, C. Duran, R. Serrano, M. Sarmiento, K.-D. Nguyen, A. Tsukamoto, K. Suzaki, and C.-K. Pham, "Trusted execution environment hardware by isolated heterogeneous architecture for key scheduling," *IEEE Access*, vol. 10, pp. 46014–46027, 2022, doi: 10.1109/ACCESS.2022.3169767.
[35] R. Maes, *Physically Unclonable Functions*. Cham, Switzerland: Springer, Aug. 2016.
[36] G. Coker, J. Guttman, P. Loscocco, A. Herzog, J. Millen, B. O'Hanlon, J. Ramsdell, A. Segall, J. Sheehy, and B. Sniffen, "Principles of remote attestation," *Int. J. Inf. Secur.*, vol. 10, no. 2, pp. 63–81, Apr. 2011, doi: 10.1007/s10207-011-0124-7.
[37] *Keystone Documentation—Remote Attestation*. Accessed: Jul. 1, 2025. [Online]. Available: https://docs.keystone-enclave.org/en/latest/Getting-Started/Tutorials/Remote-Attestation.html
[38] *IEEE Standard for Local and Metropolitan Area Networks—Secure Device Identity*, IEEE Standard 802.1AR-2018, IEEE 802.1 Work. group, Jun. 2018, pp. 1–73, doi: 10.1109/IEEESTD.2018.8423794.
[39] Trusted Firmware. *Mbed TLS*. Accessed: Jul. 1, 2025. [Online]. Available: https://www.trustedfirmware.org/projects/mbed-tls/
[40] Trusted Computing Group. *DICE Attestation Architecture*. Accessed: Jul. 1, 2025. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/TCGDICEAttestationArchitecturer2202dec2020.pdf
[41] S. Sánchez-Solano, L. F. Rojas-Muñoz, M. C. Martínez-Rodríguez, and P. Brox, "Hardware-efficient configurable ring-oscillator-based physical unclonable function/true random number generator module for secure key management," *Sensors*, vol. 24, no. 17, p. 5674, Aug. 2024, doi: 10.3390/s24175674.
[42] C. Baumgartner, M. Wiseman, T. Laffey, G. Proudler, H. Birkholz, D. Challener, M. Eckel, G. Fedorkow, K. Goldman, B. Jacobs, S. Leicht, S. Potter, B. Sulzen, L. Jacquin, N. Smith, C. Fenner, D. Rutigliano, and B. Weeks. (2021). *TPM 2.0 Keys for Device Identity and Attestation*. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/TPM-2p0-Keys-for-Device-Identity-and-Attestationv1r12pub10082021.pdf
[43] Xilinx. (2017). *Genesys 2 FPGA Board Reference Manual*. [Online]. Available: https://digilent.com/reference/media/reference/programmable-logic/genesys-2/genesys2rm.pdf
[44] P. Navarro-Torrero, M. C. Martínez-Rodríguez, Á. Barriga-Barros, and P. Brox, "Full open-source implementation of an academic RISC-V on FPGA," in *Proc. XVI Congreso de Tecnología, Aprendizaje y Enseñanza de la Electrónica (TAEE)*, Jun. 2024, pp. 1–5, doi: 10.1109/TAEE59541.2024.10604934.



ENRICO BRAVI (Student Member, IEEE) received the M.Sc. degree in computer engineering (cybersecurity) from Politecnico di Torino. He is currently pursuing the Ph.D. degree in computer engineering. He is a member with the TORSEC Cybersecurity Research Group, Politecnico di Torino. His current research interests include trusted computing, trusted execution environments, and remote attestation.



SILVIA SISINNI (Member, IEEE) received the M.Sc. degree in computer engineering from Politecnico di Torino. She is currently pursuing the Ph.D. degree in computer engineering. She is a member with the TORSEC Cybersecurity Research Group. Her current research interests include trusted execution environments, trusted computing, trusted channels, and confidential computing.



LORENZO FERRO received the M.Sc. degree in computer engineering (cybersecurity) from Politecnico di Torino, where he is currently pursuing the Ph.D. degree in computer engineering. He is also a member with the TORSEC Cybersecurity Research Group, Politecnico di Torino. His research interests include trusted computing, trusted execution environments, and remote attestation.



VALERIO DONNINI received the M.Sc. degree in computer engineering (cybersecurity) from Politecnico di Torino. He specializes in automotive cybersecurity and compliance with R155 and ISO/SAE 21434 regulations. Previously, he was a Research Fellow with Politecnico di Torino, working with the TORSEC Cybersecurity Research Group. His research interests include threat analysis, low-level programming, and security design.



ANTONIO LIOY received the M.Sc. degree (summa cum laude) in electronic engineering and the Ph.D. degree in computer engineering from Politecnico di Torino. He is currently a Full Professor with Politecnico di Torino, where he leads the TORSEC Cybersecurity Research Group. His current research interests include network security, policy-based system protection, trusted computing, and electronic identity.